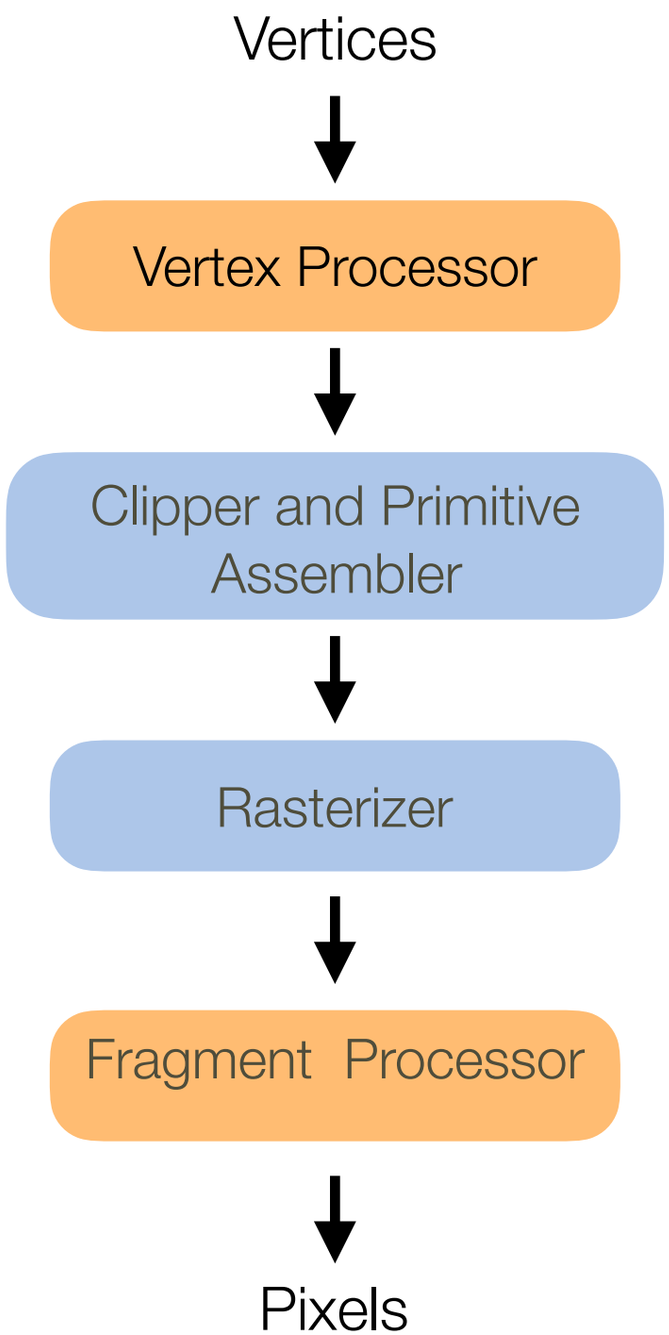
BCA611 Video Oyunları için 3B Grafik

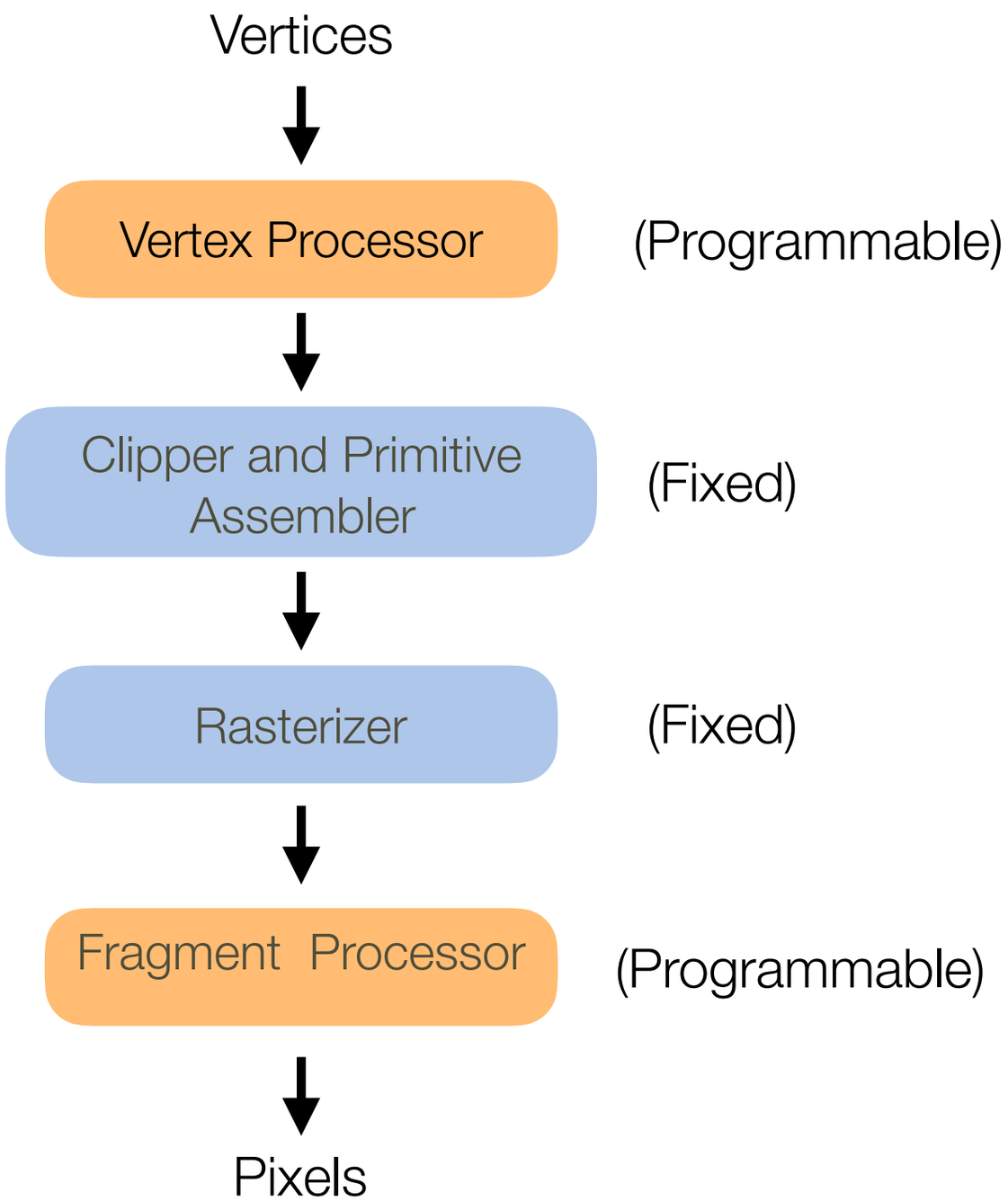
Ders3 - Shader Programming with WebGL

Zümra Kavafoğlu

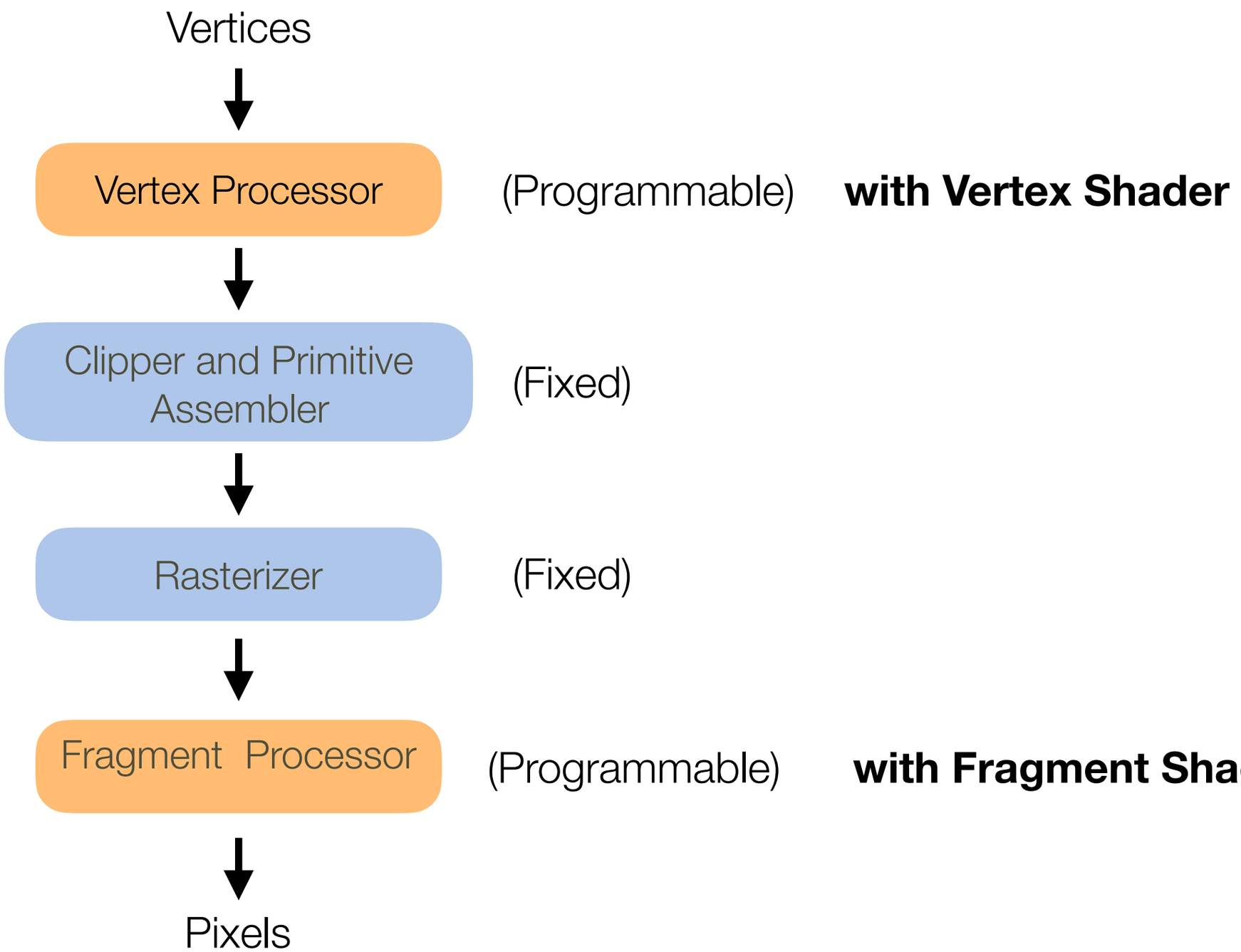
Recall the rendering pipeline



Recall the rendering pipeline



Recall the rendering pipeline



What is a shader?

Shaders are little programs that execute on the GPU

```
attribute vec3 aVertexPosition;
varying highp vec4 vColor;
void main(void) {
    gl_Position = vec4(aVertexPosition, 1.0);
    if(gl_Position.x < 0.0)
        vColor = vec4(1.0, 0.0, 0.0, 1.0);
    else
        vColor = vec4(0.0, 1.0, 0.0, 1.0);
}
```

They include variables and a main function.

What is a shader?

Shaders are little programs that execute on the GPU

- They can be seen as functions transforming inputs into outputs
- They communicate with application and each other via these inputs and outputs

What is a shader?

Shaders are little programs that execute on the GPU

- They can be seen as functions transforming inputs into outputs
- They communicate with application and each other via these inputs and outputs

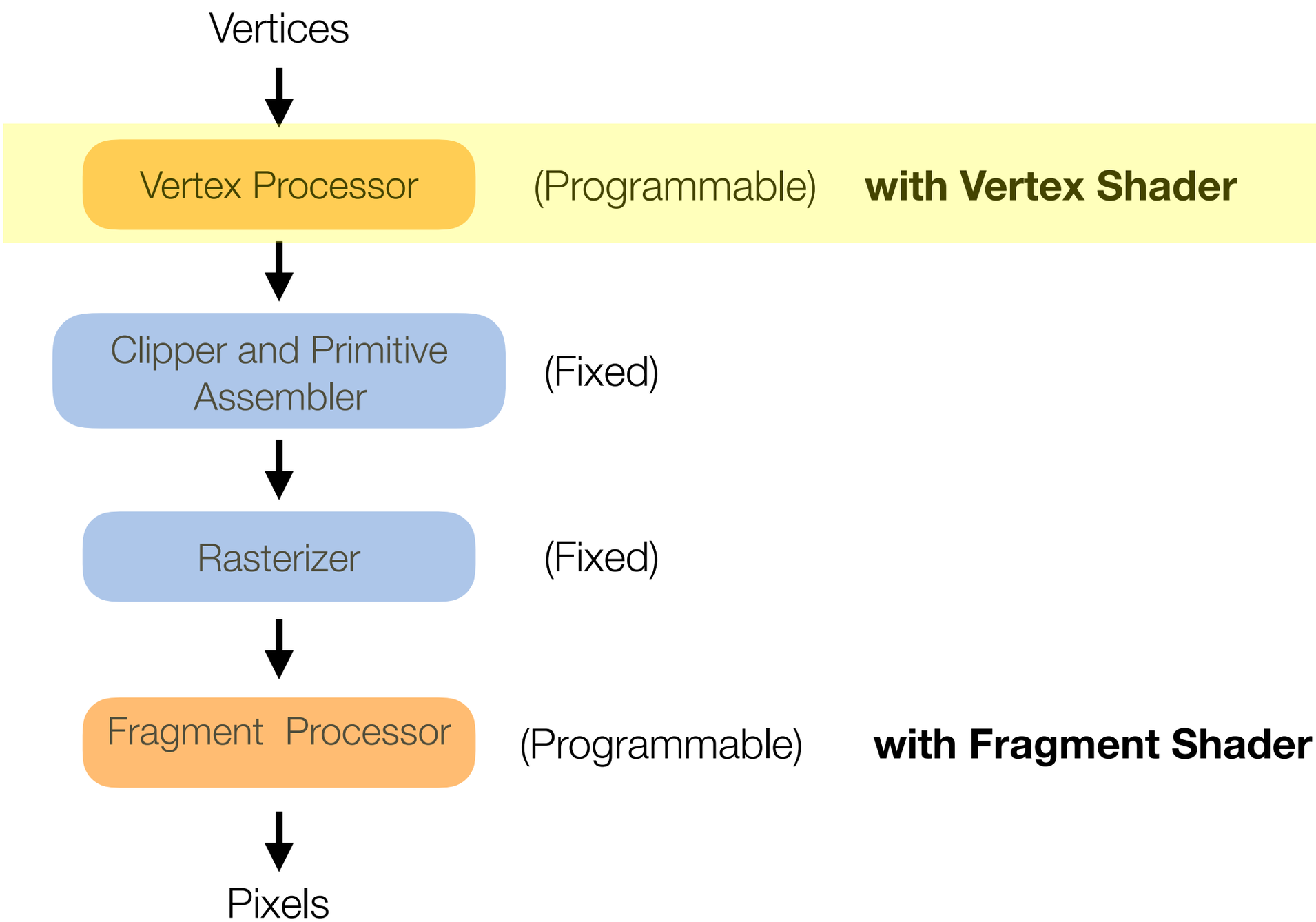
Shaders are written with GLSL(Graphics Library Shader Language) - a C like language

WebGL shaders

- WebGL requires two shaders for rendering: Vertex Shader and Fragment Shader

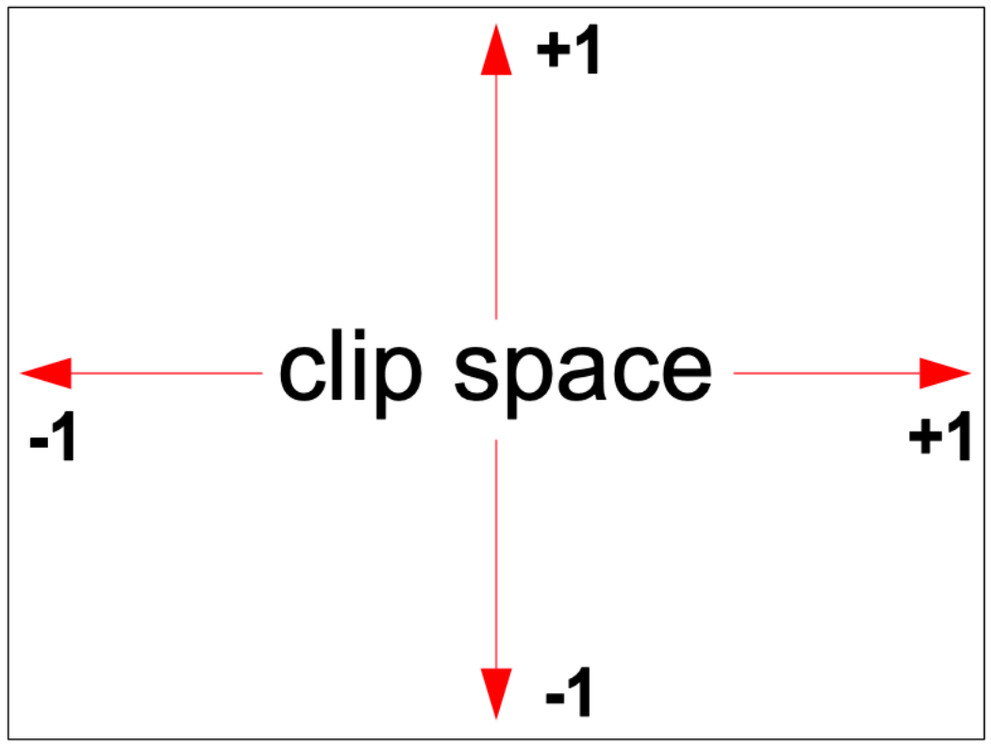
- Vertex shader and fragment shader are linked together into a shader program

Vertex Shader



Vertex Shader

Main job : to generate clipspace coordinates



Vertex Shader

Main job : to generate clipspace coordinates

- It’s called once per vertex
- Each time it’s called the special built-in global variable gl_Position should be set to the proper clipspace coordinates

```
void main( ){

   gl_Position = clipSpaceCoordinates

}
```

Built-in variables: gl_Position

The OpenGL Shading Language defines a number of special variables for the various shader stages. These **built-in variables** (or built-in variables) have special properties. They are usually for communicating with certain fixed-functionality. By convention, all predefined variables start with "gl_"; no user-defined variables may start with this.*

gl_Position — contains the clip-space position of the current vertex in homogenous coordinates (last component 1)

*https://www.khronos.org/opengl/wiki/Built-in_Variable_(GLSL)

Vertex Shader

Vertex shaders work with three types of data:

- Attributes
- Uniforms
- Varyings

Vertex Shader

Vertex shaders work with three types of data:

- Attributes
- Uniforms
- Varyings

data taken from application

Vertex Shader Data Types

Vertex shaders work with three types of data:

- Attributes
- Uniforms
- Varyings

data passed to fragment shader

Vertex Attributes

- Vertex attributes are used to communicate from "outside" to the vertex shader.
- Values are provided per vertex (and not globally for all vertices).
- Attributes **can't** be defined in the fragment shader.

*https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/attributes.php

Vertex Attributes

- Vertex attributes are used to communicate from "outside" to the vertex shader.

- Values are provided per vertex (and not globally for all vertices).

- Attributes **can't** be defined in the fragment shader.

Give examples of possible attributes?

Hint: they should be values per vertex

*https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/attributes.php

Vertex Attributes

Attributes can be defined in the vertex shader using the "attribute" qualifier*:

```
attribute vec3 aVertexPosition;
```

attribute qualifier

attribute type

attribute name

*https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/attributes.php

Attribute types

Attributes can use `float`, `vec2`, `vec3`, `vec4`, `mat2`, `mat3`, and `mat4` as types.

*https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/attributes.php

Vertex Attribute

Vertex shader:

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

Javascript program

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 3);
}
```

SingleTriangle.html

Vertex Attribute

Vertex shader:

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

Javascript program

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 3);
}
```

returns the location of the attribute named "aVertexPosition"

SingleTriangle.html

Vertex Attribute

Vertex shader:

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

Javascript program

attributes cannot be used unless enabled, and are disabled by default, you need to call `enableVertexAttribArray()` to enable individual attributes so that they can be used.

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 3);
}
```

SingleTriangle.html

Vertex Attribute

Vertex shader:

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

Javascript program

set triangleVertexPositionBuffer as the active buffer

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 3);
}
```

SingleTriangle.html

Vertex Attribute

Vertex shader:

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

Javascript program

binds the active gl.ARRAY_BUFFER to the attribute located at vertexPositionAttribute and specifies its layout

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 3);
}
```

SingleTriangle.html

Vertex Attribute

Vertex shader:

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

Javascript program

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 3);
}
```

number of components in the array buffer per vertex attribute(must be 1, 2, 3 or 4)

SingleTriangle.html

Vertex Attribute

Vertex shader:

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>
```

Javascript program

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 3);
}
```

data type of each component

SingleTriangle.html

Uniforms

Uniform variables are used to communicate with your vertex or fragment shader from "outside". Uniform variables are **read-only** and have the same value among all processed vertices/ fragments. You can only change them within your javascript program*

```
uniform vec3 uOffset;
```

uniform qualifier

uniform type

uniform name

*https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/uniform.php

Example : translate all vertices one unit right by using uniforms

---

1) Define a uniform in the vertex shader and add it to vertex position.

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    uniform vec3 uOffset;
    void main(void) {
        gl_Position = vec4(aVertexPosition + uOffset, 1.0);
    }
</script>
```

Example : translate all vertices one unit right by using uniforms

---

1) Define a uniform in the vertex shader and add it to vertex position.

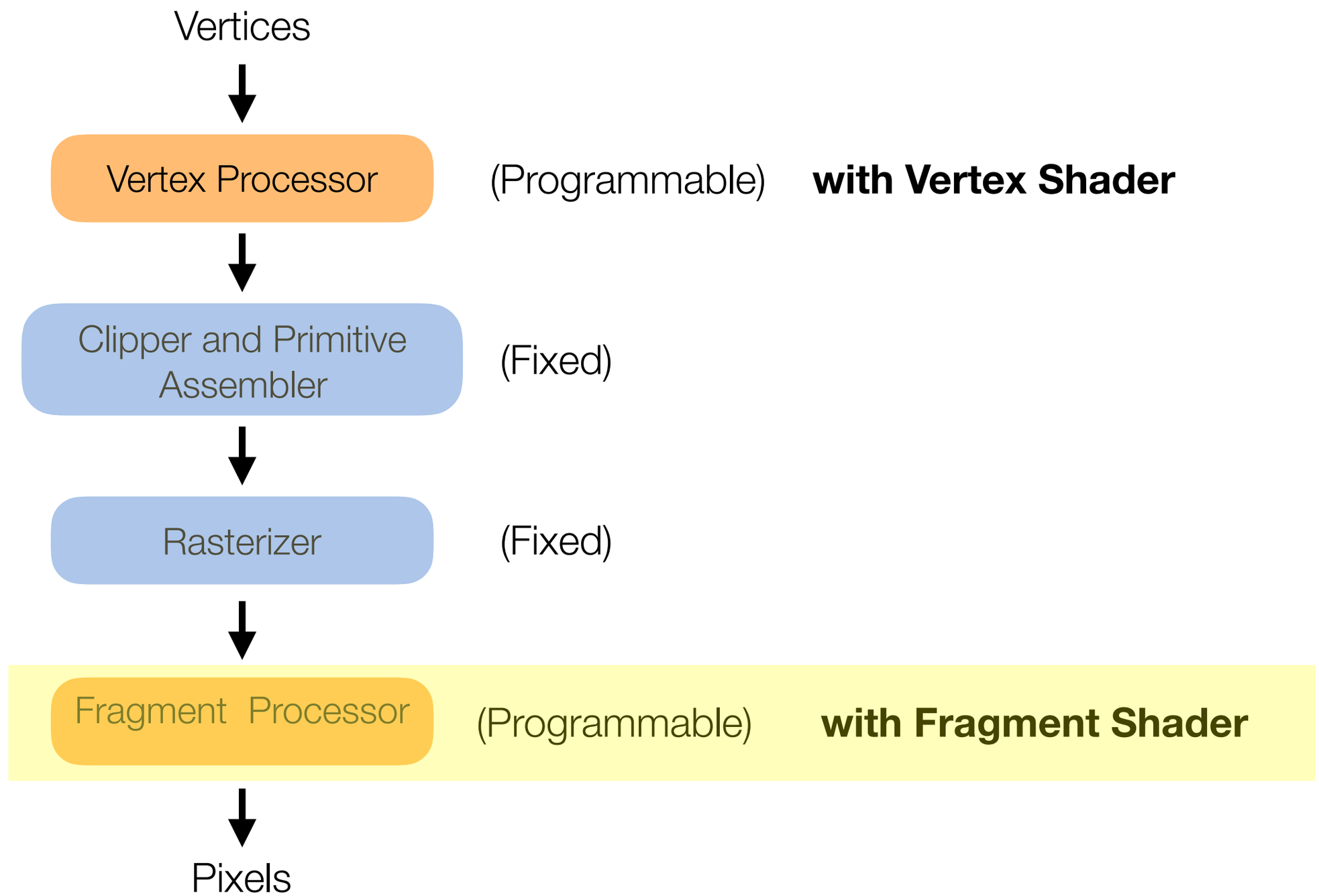
```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    uniform vec3 uOffset;
    void main(void) {
        gl_Position = vec4(aVertexPosition + uOffset, 1.0);
    }
</script>
```

2) Javascript program: get the location of uniform and set the desired value

```
uOffsetLoc = gl.getUniformLocation(glProgram, "uOffset");
gl.uniform3fv(uOffsetLoc, [1.0,0,0]);
```

SingleTriangleTranslated.html

Fragment Shader



Fragment Shader

Main job : to provide a color for the fragments

Fragment Shader

Main job : to provide a color for the pixels

- It’s called once per pixel
- Each time it’s called the special built-in variable gl_FragColor should be set to some color

```
void main( ){

   gl_FragColor = pixelColor

}
```

Example: set each fragment color to white in fragment shader

Example: set each fragment color to white in fragment shader

Fragment shader

```
<script id="shader-fs" type="x-shader/x-fragment">
    void main(void) {
       gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
</script>
```

Fragment Shader Data Types

Fragment shaders work with two types of data:

- Uniforms
- Varyings
-

Fragment Shader Data Types

Fragment shaders work with two types of data:

- Uniforms

data taken from application

- Varyings

Fragment Shader Data Types

Fragment shaders work with two types of data:

- Uniforms
- Varyings

data passed from vertex shader

Varyings

Most of the time fragment shaders need data from vertices. This data is interpolated through frames automatically in fragment shader. This is why it’s called varying.

Example : Color the triangle with vertex colors by using attributes and varyings

Example : Color the triangle with vertex colors by using attributes and varyings

---

1) Vertex Shader: Define an attribute for passing color data from application for each vertex

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;
    varying highp vec4 vColor;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

Example : Color the triangle with vertex colors by using attributes and varyings

---

2) Vertex Shader: Define a varying that will pass vertex color data to fragment shader

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;
    varying highp vec4 vColor;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

```
varying highp vec4 vColor;
```

qualifier precision type name

Example : Color the triangle with vertex colors by using attributes and varyings

---

3) Vertex Shader: Set the value of varying by using color attribute

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;
    varying highp vec4 vColor;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

Example : Color the triangle with vertex colors by using attributes and varyings

---

4) Fragment Shader: Define the varying with the same type, precision and name in fragment shader

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;
    varying highp vec4 vColor;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

```
<script id="shader-fs" type="x-shader/x-fragment">
    varying highp vec4 vColor;
    void main(void) {
       gl_FragColor = vColor;
    }
</script>
```

Example : Color the triangle with vertex colors by using attributes and varyings

---

4) Fragment Shader: set gl_FragColor with the value of varying

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;
    varying highp vec4 vColor;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

```
<script id="shader-fs" type="x-shader/x-fragment">
    varying highp vec4 vColor;
    void main(void) {
        gl_FragColor = vColor;
    }
</script>
```

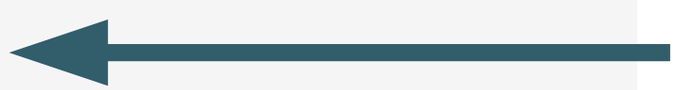
Example : Color the triangle with vertex colors by using attributes and varyings

---

4) Fragment Shader: set gl_FragColor with the value of varying

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;
    varying highp vec4 vColor;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

```
<script id="shader-fs" type="x-shader/x-fragment">
    varying highp vec4 vColor;
    void main(void) {
        gl_FragColor = vColor;
    }
</script>
```



here interpolated vertex color values are set

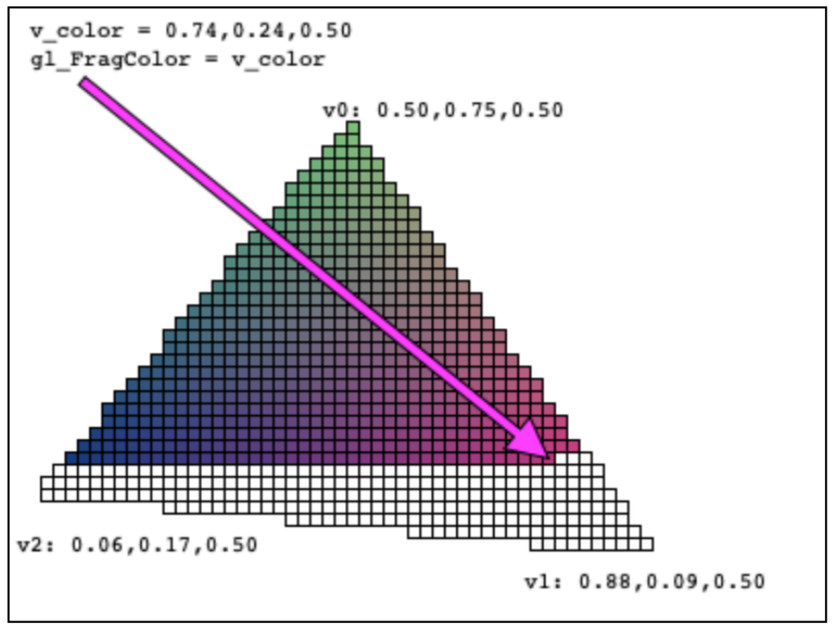
Example : Color the triangle with vertex colors by using attributes and varyings

---

4) Fragment Shader: set gl_FragColor with the value of varying

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;
    varying highp vec4 vColor;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

```
<script id="shader-fs" type="x-shader/x-fragment">
    varying highp vec4 vColor;
    void main(void) {
        gl_FragColor = vColor;
    }
</script>
```



here interpolated vertex color values are set

Example : Color the triangle with vertex colors by using attributes and varyings

---

We should supply values for the aVertexColor attribute of vertex shader. First create and setup buffer from the desired color values(RGB) for each vertex.

```
function setupBuffers()
{
    var triangleVertices = [
        -0.5, -0.5, 0.0,
        0.5, -0.5, 0.0,
        0.0, 0.0, 0.0,
    ];

    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertices), gl.STATIC_DRAW);

    var triangleVertexColors = [
        0.06, 0.17, 0.5,
        0.88, 0.09, 0.5,
        0.5, 0.75, 0.5,
    ];
    triangleColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleColorBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertexColors), gl.STATIC_DRAW);

}
```

Example : Color the triangle with vertex colors by using attributes and varyings

---

Then communicate with the attribute and specify the way vertex shader takes the values from the buffer.

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);

    vertexColorAttribute = gl.getAttribLocation(glProgram, "aVertexColor");
    gl.enableVertexAttribArray(vertexColorAttribute);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleColorBuffer);
    gl.vertexAttribPointer(vertexColorAttribute, 3, gl.FLOAT, false, 0, 0);

    gl.drawArrays(gl.TRIANGLES, 0, 3);
}
```

Example : Color the triangle with vertex colors by using attributes and varyings

SingleTriangleColored.html

Uniforms

Works the same way as for vertex shaders

Example: Darken or lighten the color of each fragment with the same amount by using uniforms

---

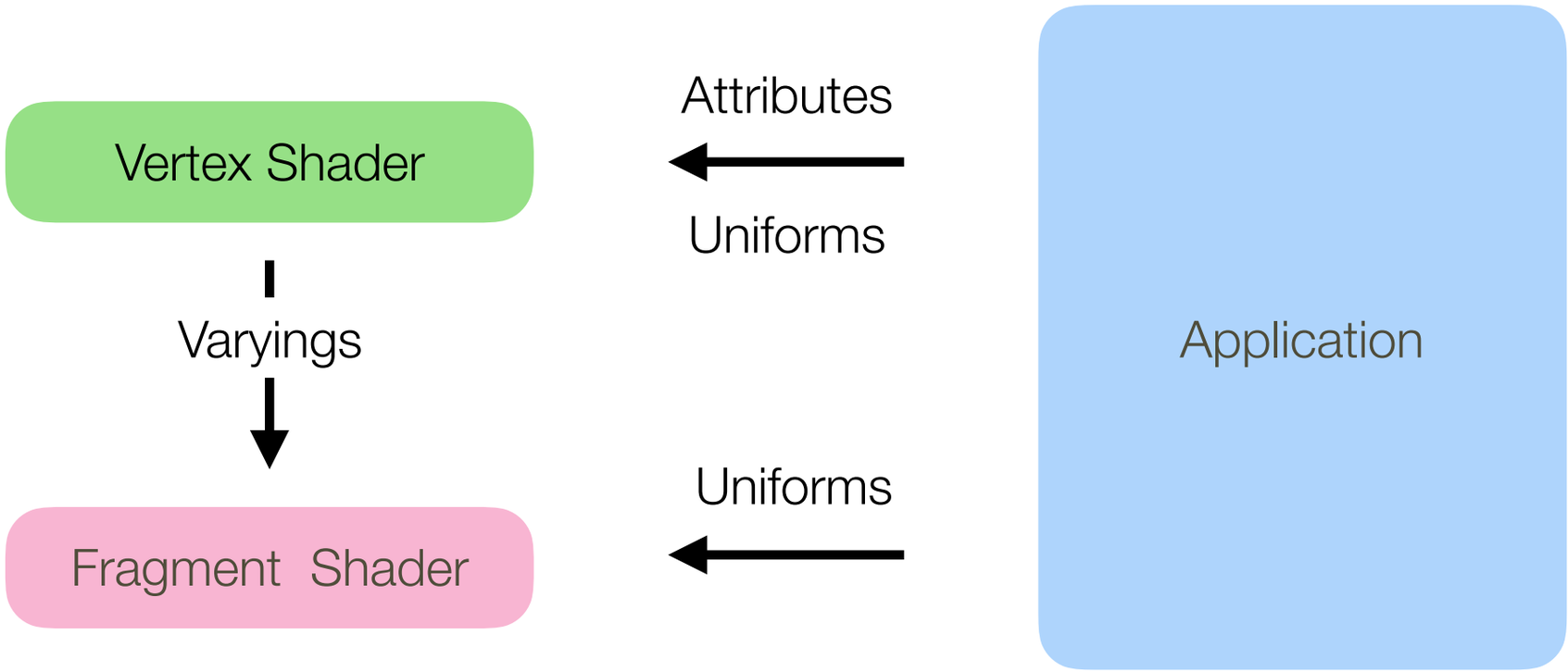
1) Fragment Shader: Define a uniform and multiply it with vColor varying

```
<script id="shader-fs" type="x-shader/x-fragment">
    varying highp vec4 vColor;
    uniform mediump float uColorFactor;
    void main(void) {
       gl_FragColor = vColor * uColorFactor;
    }
</script>
```

2) Javascript program: get the location of uniform and set the desired value

```
uColorFactorLoc = gl.getUniformLocation(glProgram, "uColorFactor");
gl.uniform1f(uColorFactorLoc, 0.4);
```

Variable Traffic



Creating, compiling and linking shaders

initWebGL()

```
var gl = null,
    canvas = null,
    glProgram = null,
    fragmentShader = null,
    vertexShader = null;

function initWebGL()
{
    canvas = document.getElementById("my-canvas");
    try{
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
    }catch(e){
    }

    if(gl)
    {
        initShaders();

        window.init();

        //just call once to start updating
        loop();

    }else{
        alert(  "Error: Your browser does not appear to support WebGL.");
    }
}
```

get the canvas with id “my-canvas”

initWebGL()

```
var gl = null,
    canvas = null,
    glProgram = null,
    fragmentShader = null,
    vertexShader = null;

function initWebGL()
{
    canvas = document.getElementById("my-canvas");
    try{
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
    }catch(e){
    }

    if(gl)
    {
        initShaders();

        window.init();

        //just call once to start updating
        loop();

    }else{
        alert(  "Error: Your browser does not appear to support WebGL.");
    }
}
```

get webGL context if it's supported by the browser

initWebGL()

```
var gl = null,
    canvas = null,
    glProgram = null,
    fragmentShader = null,
    vertexShader = null;

function initWebGL()
{
    canvas = document.getElementById("my-canvas");
    try{
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
    }catch(e){
    }

    if(gl)
    {
        initShaders();

        window.init();

        //just call once to start updating
        loop();

    }else{
        alert(  "Error: Your browser does not appear to support WebGL.");
    }
}
```

if your browser supports webgl

initWebGL()

```
var gl = null,
    canvas = null,
    glProgram = null,
    fragmentShader = null,
    vertexShader = null;

function initWebGL()
{
    canvas = document.getElementById("my-canvas");
    try{
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
    }catch(e){
    }

    if(gl)
    {
        initShaders();

        window.init();

        //just call once to start updating
        loop();

    }else{
        alert(  "Error: Your browser does not appear to support WebGL.");
    }
}
```

We have two parts in the program, the application written in javaScript and the shaders written in GLSL. initShaders compile shaders and link application and shaders to each other.

initShaders()

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

initShaders()

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShad
    gl.attachShader(glProgram, fragmentSh

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram
        alert("Unable to initialize the s
    }

    //use program
    gl.useProgram(glProgram);
}
```

Fragment shader script

```
<script id="shader-fs" type="x-shader/x-fragment">
    varying highp vec4 vColor;

    void main(void) {
       gl_FragColor = vColor;
    }
</script>
```

initShaders()

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader                              l.FRAGMENT_SHADER);

    //create progra
    glProgram = gl

    //attach shade
    gl.attachShade
    gl.attachShade

    gl.linkProgram

    if (!gl.getProg
        alert("Unab
    }

    //use program
    gl.useProgram(g
}
```

Vertex shader script

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying highp vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

initShaders()

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = g

    //attach shad
    gl.attachShad
    gl.attachShad

    gl.linkProgra

    if (!gl.getPr
        alert("Un
    }

    //use progra
    gl.useProgra
}
```

```
function makeShader(src, type)
{
    //compile the vertex shader
    var shader = gl.createShader(type);
    gl.shaderSource(shader, src);
    gl.compileShader(shader);

    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        alert("Error compiling shader: " + gl.getShaderInfoLog(shader));
    }
    return shader;
}
```

initShaders()

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

compile vertex shader

initShaders(): compile shaders

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

compile fragment shader

initShaders(): WebGLProgram

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

create and initialize a WebGLProgram object

initShaders(): WebGLProgram

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

The WebGLProgram is part of the WebGL API and is a combination of two compiled `WebGLShader`s consisting of a vertex shader and a fragment shader (both written in GLSL).

initShaders(): WebGLProgram

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

if shader compilation is successful, the application and shaders can be linked together

initShaders(): WebGLProgram

```
function initShaders()
{
    //get shader source
    var fs_source = document.getElementById('shader-fs').innerHTML,
        vs_source = document.getElementById('shader-vs').innerHTML;

    //compile shaders
    vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);

    //create program
    glProgram = gl.createProgram();

    //attach shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);

    gl.linkProgram(glProgram);

    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }

    //use program
    gl.useProgram(glProgram);
}
```

tell GPU to use the program

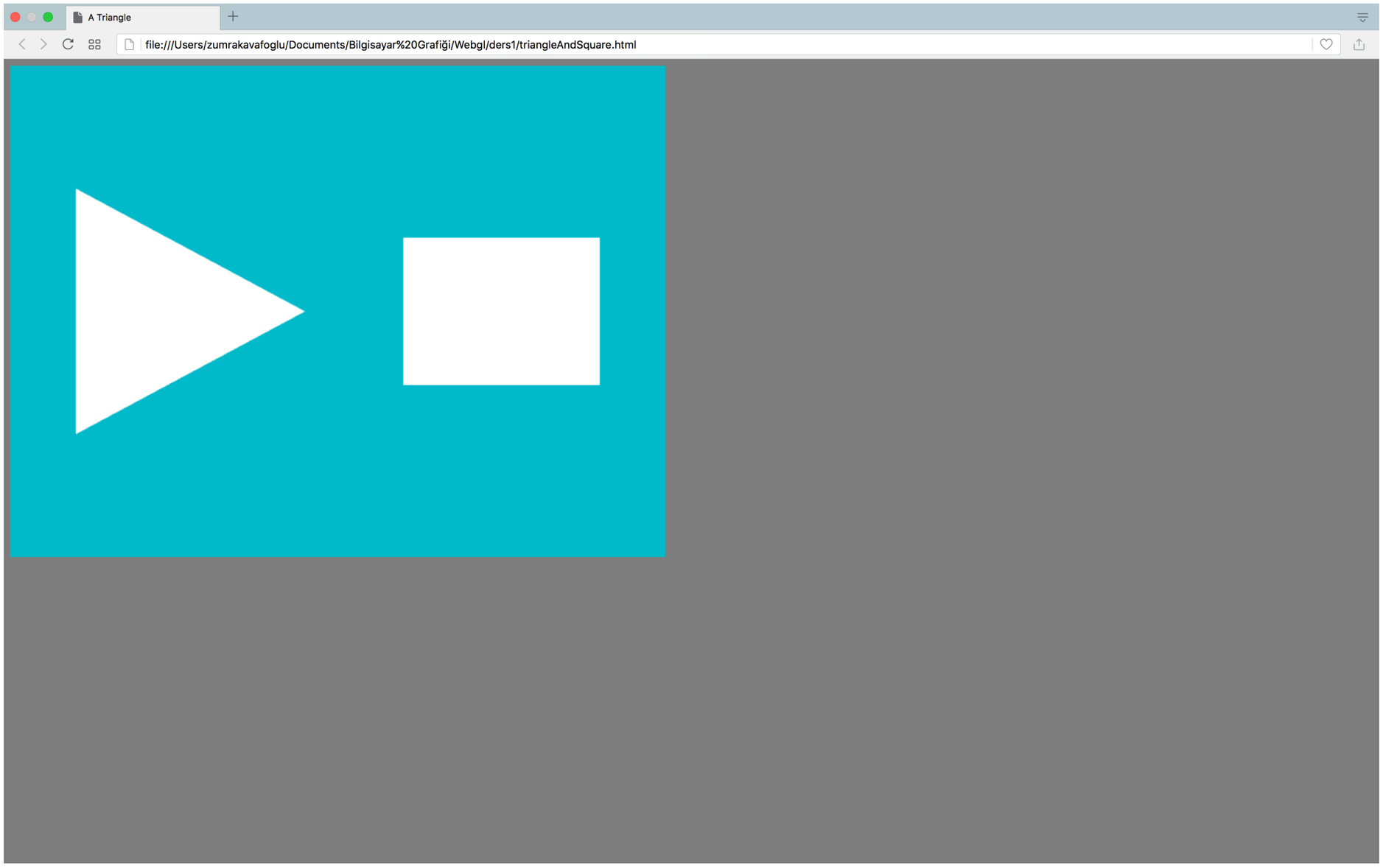
For more

See WebGL 1.0 Quick reference guide

https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf

Examples

Bir üçgen ve bir kare çizdirme



TriangleAndSquare.html

Bir üçgen ve bir karenin köşe pozisyonları için VBO oluşturma

```
function setupBuffers()
{
    var triangleVertices = [
        -0.8, 0.5, 0.0,
         -0.8, -0.5, 0.0,
        -0.1, 0.0, 0.0,
    ];

    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertices), gl.STATIC_DRAW);

    var squareVertices = [
         0.2,  0.3,  0.0,
         0.2,  -0.3,  0.0,
         0.8, 0.3,  0.0,
         0.8, -0.3,  0.0
    ];
    squareVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);

    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(squareVertices), gl.STATIC_DRAW);

}
```

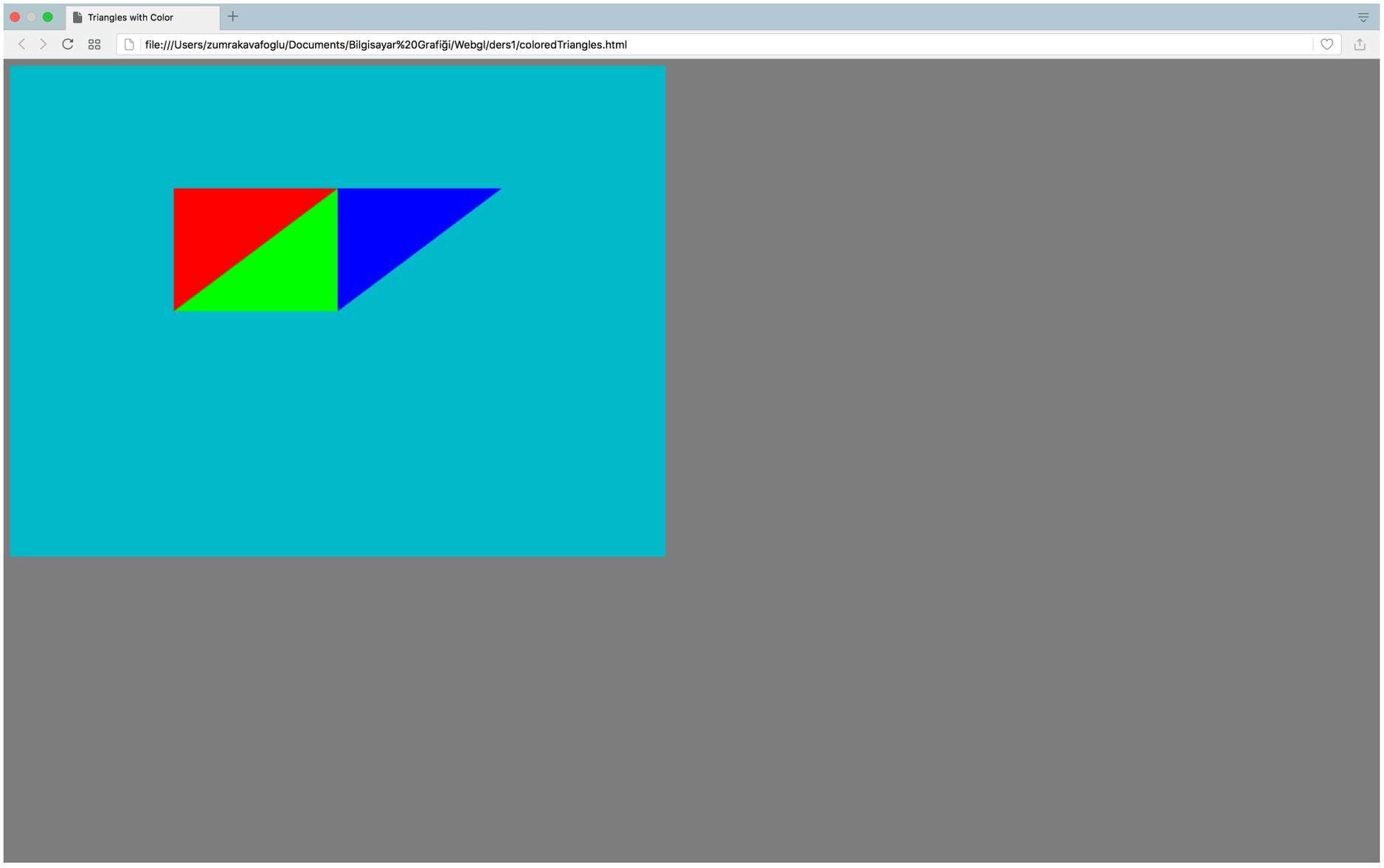
Üçgen ve kareyi çizdirme

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 3);

    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
}
```

Farklı renkte bitişik üçgenler oluşturma



ColoredTriangles.html

Farklı renkte bitişik üçgenler oluşturma

```
function setupBuffers()
{
    var triangleVertices = [
        -0.5, 0.5, 0.0,
         -0.5, 0.0, 0.0,
        0.0, 0.5, 0.0,

        -0.5, 0.0, 0.0,
        0.0, 0.5, 0.0,
        0.0, 0.0, 0.0,

        0.0, 0.5, 0.0,
        0.0, 0.0, 0.0,
        0.5, 0.5, 0.0,
    ];

    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertices), gl.STATIC_DRAW);

    var triangleVertexColors = [
        1.0, 0.0, 0.0,
        1.0, 0.0, 0.0,
        1.0, 0.0, 0.0,

        0.0, 1.0, 0.0,
        0.0, 1.0, 0.0,
        0.0, 1.0, 0.0,

        0.0, 0.0, 1.0,
        0.0, 0.0, 1.0,
        0.0, 0.0, 1.0,
    ];
    triangleColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleColorBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertexColors), gl.STATIC_DRAW);

}
```

Farklı renkte bitişik üçgenler oluşturma

```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);

    vertexColorAttribute = gl.getAttribLocation(glProgram, "aVertexColor");
    gl.enableVertexAttribArray(vertexColorAttribute);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleColorBuffer);
    gl.vertexAttribPointer(vertexColorAttribute, 3, gl.FLOAT, false, 0, 0);

    gl.drawArrays(gl.TRIANGLES, 0, 9);
}
```

Index Buffer kullanarak bitişik üçgenler çizdirme

```
function setupBuffers()
{
    var triangleVertices = [
        -0.5, 0.5, 0.0,
         -0.5, 0.0, 0.0,
        0.0, 0.5, 0.0,
        0.0, 0.0, 0.0,
        0.5, 0.5, 0.0,
    ];

    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertices), gl.STATIC_DRAW);

    var triangleVertexColors = [
        1.0, 0.0, 0.0,
        1.0, 0.0, 0.0,
        1.0, 0.0, 0.0,
        0.0, 1.0, 0.0,
        0.0, 0.0, 1.0,
    ];
    triangleColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleColorBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertexColors), gl.STATIC_DRAW);

    var triangleVertexIndices = [
        //front face
        0,1,2,
        1,3,2,
        2,3,4
    ];

    triangleVerticesIndexBuffer = gl.createBuffer();
    triangleVerticesIndexBuffer.number_vertex_points = triangleVertexIndices.length;
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, triangleVerticesIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(triangleVertexIndices), gl.STATIC_DRAW);
}
```

Ortak köşelerin pozisyon ve renk değerleri yalnızca bir kere yazılıyor

Index Buffer kullanarak bitişik üçgenler çizdirme

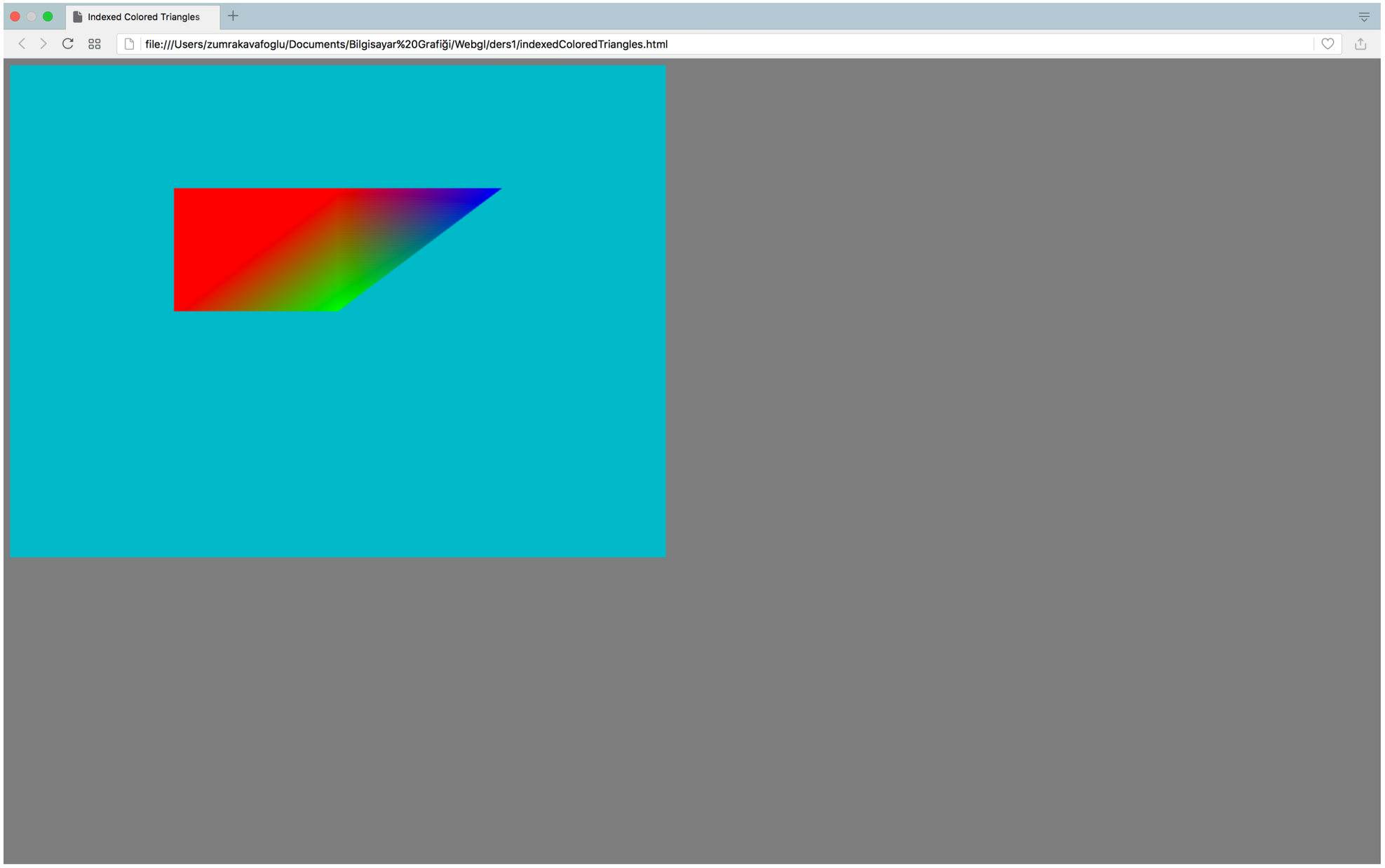
```
function drawScene()
{
    vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
    gl.enableVertexAttribArray(vertexPositionAttribute);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);

    vertexColorAttribute = gl.getAttribLocation(glProgram, "aVertexColor");
    gl.enableVertexAttribArray(vertexColorAttribute);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleColorBuffer);
    gl.vertexAttribPointer(vertexColorAttribute, 3, gl.FLOAT, false, 0, 0);
```

```
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, triangleVerticesIndexBuffer);
    gl.drawElements(gl.TRIANGLES, triangleVerticesIndexBuffer.number_vertex_points, gl.UNSIGNED_SHORT, 0);
```

```
}
```

Index Buffer kullanarak bitişik üçgenler çizdirme

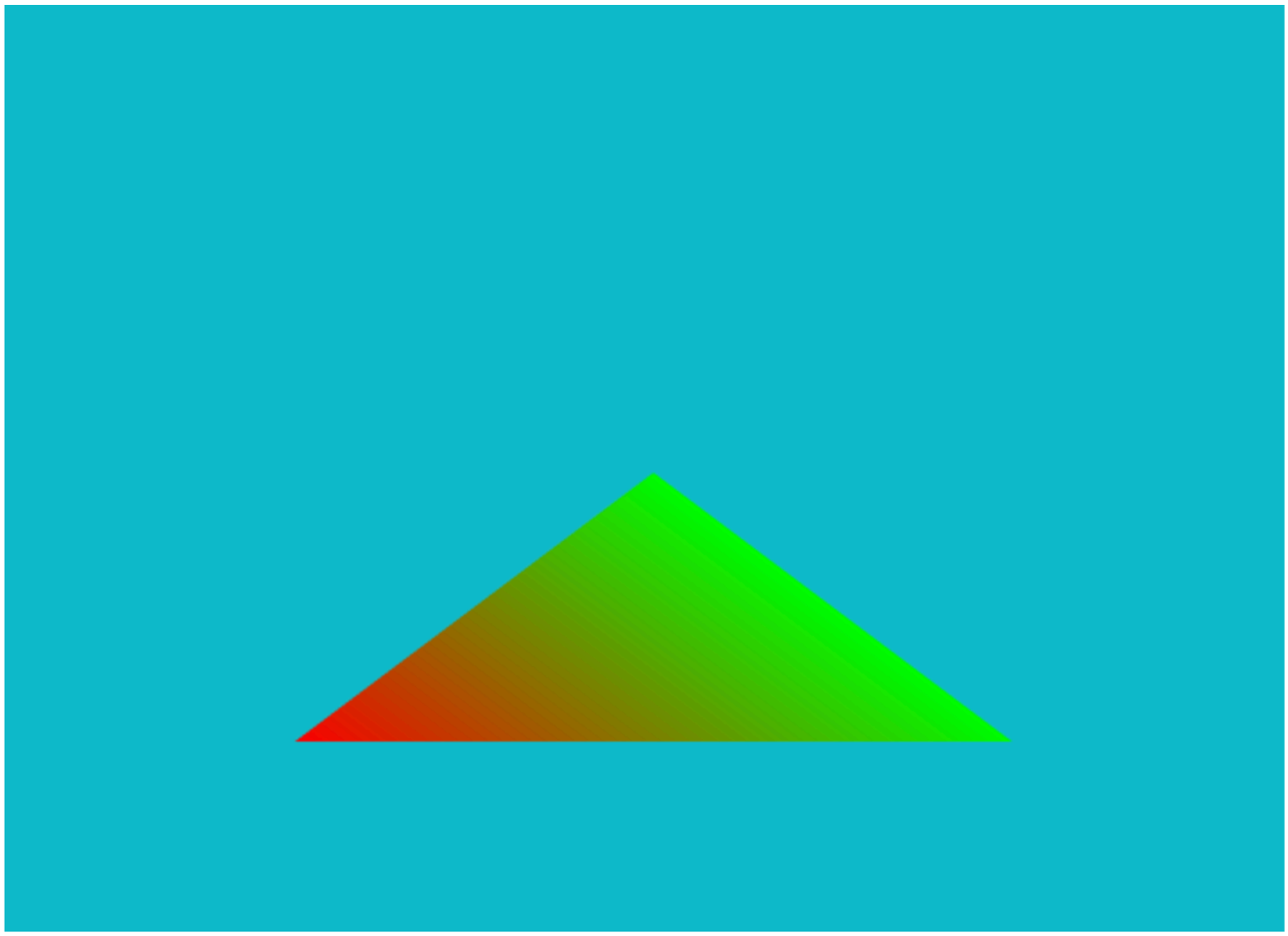


IndexedColoredTriangles.html

Köşe pozisyonuna göre köşeleri renklendirme (Shader kodunda if-else kullanma)

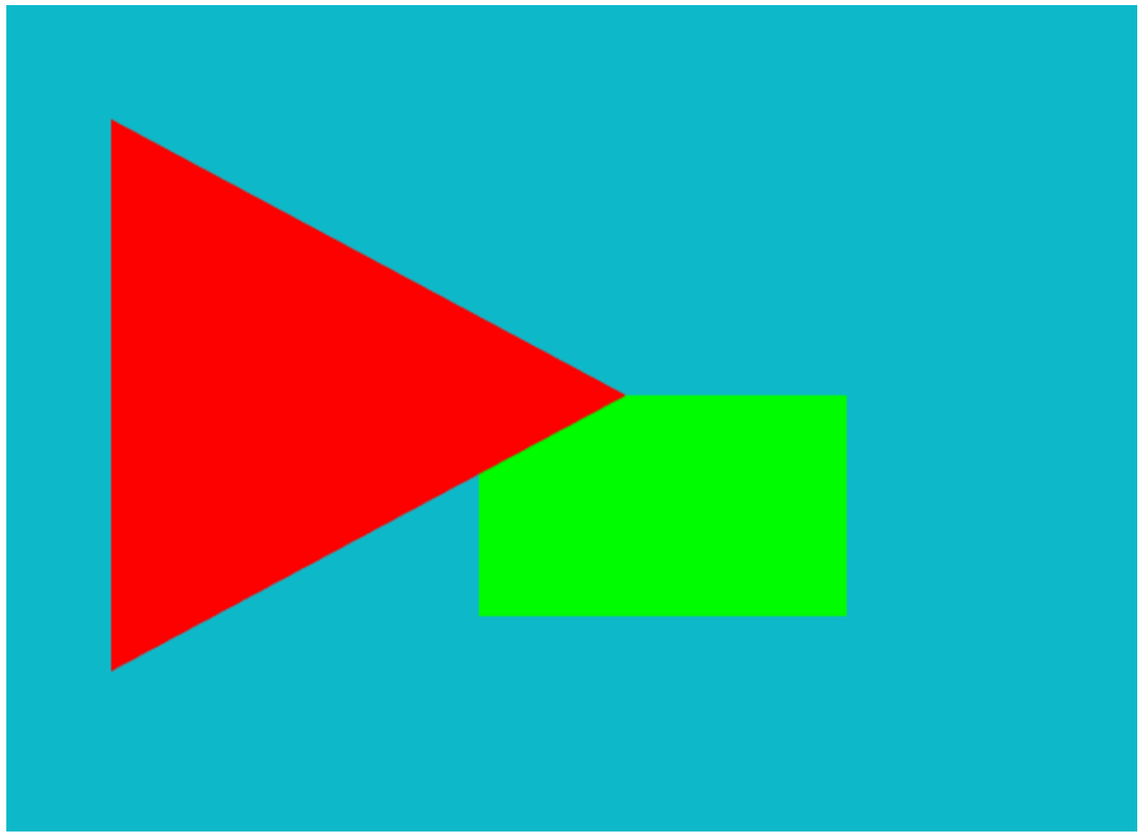
```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    varying highp vec4 vColor;
    void main(void) {
        gl_Position = vec4(aVertexPosition, 1.0);
        if(gl_Position.x < 0.0)
            vColor = vec4(1.0, 0.0, 0.0, 1.0);
        else
            vColor = vec4(0.0, 1.0, 0.0, 1.0);
    }
</script>
```

Köşe pozisyonuna göre köşeleri renklendirme (Shader kodunda if-else kullanma)



SingleTriangleColoredWithPosition.html

Overlapping colored triangle and square



```
function setupWebGL()
{
    //set the clear color to a shade of green
    gl.clearColor(0.08, 0.74, 0.8, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.enable(gl.DEPTH_TEST);

    gl.viewport(0, 0, canvas.width, canvas.height);
}
```

TriangleAndSquareOverlapped.html

Alıştırma

- Aşağıdaki ekran görüntüsüne benzer şekilde rasgele renklerde, büyüklüklerde ve konumlarda 50 adet dikdörtgen çizdiren bir WebGL programı yazınız.

- Dikdörtgen renklerini vertex attribute değil fragment uniform kullanarak belirleyiniz.